Summary of summer work

Tung Nguyen Quang

Last edited: September 20, 2021

1 Introduction

This report contains some technical information regarding the algorithms that were implemented in this repository, specifically the ones that act on Riemannian manifolds.
In particular, section 2 presents the data-sets that were included in the research, whereas section 3 goes into detail about the algorithms that have been used for those data-sets. Last but not least, section 4 verbally explains the running procedure of the code, and section 5 closes the report with results obtained from running the algorithms on the data-sets.
Information about how to run the codes in the repository can be found in the README file.

2 Data-sets

For this research, 3 data-sets have been found to be suitable for testing the algorithms. Readers can refer to the README file for more information on how and where to get these data-sets.

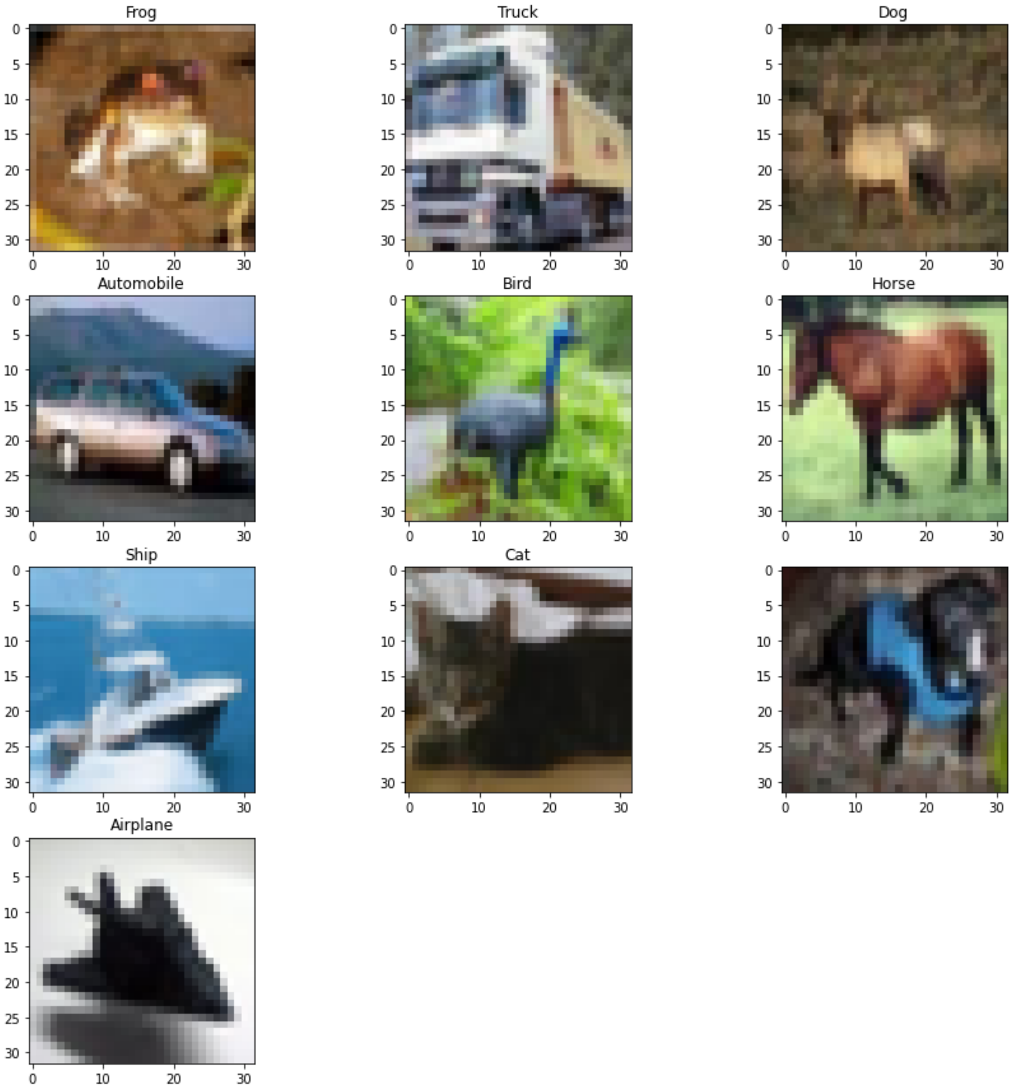
2.1 CIFAR-10

The data-set contains 60000 32x32 images of 10 classes, with 6000 images for each. Among the 10 classes, 5 belong to the animals group (cat, deer, dog, frog, horse), and 5 belong to the vehicles group (airplane, automobile, horse, ship, truck). Some example images can be found in Figure 1.
Even though half the data is not vehicle-related, a vehicle classification task would also be possible by removing all the animals data. Nevertheless, the whole data-set was still used for this research.

2.2 MIO-TCD

The data-set was used for the Traffic Surveillance Workshop and Challenge (TSWC) 2017, containing 648959 images of 11 classes, most of which are related to vehicles. Some sample images can be viewed in Figure 2.
Given the large size of the data-set, some classes have been removed for this test to lower the preprocessing and training time, particularly those that are not vehicles and one that contains too many samples. In the end, only the following classes were

Figure 1: Sample images from CIFAR data-set



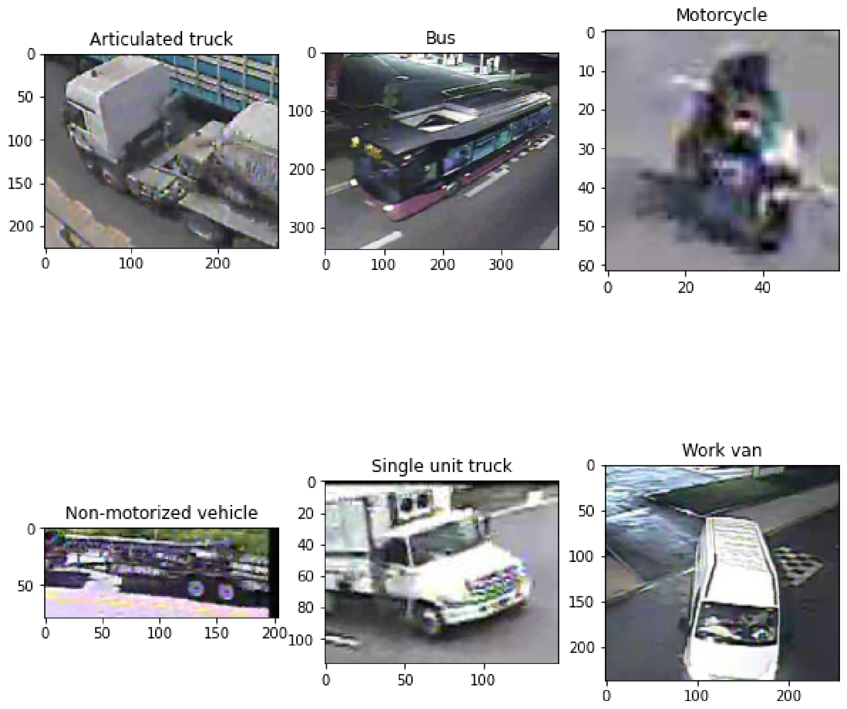
used: Articulated truck, Bus, Motorcycle, Non-motorized vehicle, Single unit truck, Work van.

2.3 TAU-Vehicle

This data-set comes from a contest during a Pattern Recognition course from Tampere Universities. It contains around 36000 images of 17 class: Ambulance, Boat, Cart, Limousine, Snowmobile, Truck, Barge, Bus, Caterpillar, Motorcycle, Tank, Van, Bicycle, Car, Helicopter, Segway, Taxi. Readers can refer to Figure 3 for sample images.

Similar to the MIO-TCD data-set, only the following classes were included: Ambulance, Bus, Car, Cart, Limousine, Motorcycle, Snowmobile, Tank, Taxi, Truck, Van.

Figure 2: Sample images from MIO-TCD data-set



3 Algortihms

This section introduces the algoriothms that have been used in the repository so far. Since the research focuses primarily on data lying on the Riemannian manifolds, only Riemannian-related algorithms will be discussed here.

3.1 Multi-class Adaboost

Adaboost is one algorithm from the Boosting [2] family, which combines a number of weak learners into one strong classifier. Though originally designed for binary classification, there have been attempts to make Adaboost work in the multi-class case, and one of them can be found in Algorithm 2 of [3].

However, for the algorithm to work with SPD matrices, small adjustments need to be made regarding the conversion of the inputs. This can be done by mapping the input data into a tangent plane on the manifold, which effectively returns the problem back to the usual vector case. To determine where the tangent plane should be, one can take the weighted mean of all the samples, and then project a tangent plane from that point, as can be seen from the algorithm below:

Figure 3: Sample images from TAU-Vehicle data-set

```
Algorithm 1: Multi-class AdaBoost on Riemannian manifolds
  Input: Training set $(\boldsymbol{X_i}, y_i)_{i=1...N}$, $\boldsymbol{X_i} \in \mathcal{M}$, $y_i \in \{0, 1, ..., K\}$
    1. Initialize weights $w_i = 1/N$, $i = 1...N$
    2. Repeat for $m = 1, 2, ..., M$:
        (a) Compute weighted mean of the points
            $\boldsymbol{\mu_m} = \arg\min_{\boldsymbol{X} \in \mathcal{M}} \sum_{i=1}^{N} w_i d^2(\boldsymbol{X_i}, \boldsymbol{X})$
        (b) Map data points to tangent space at $\boldsymbol{\mu_m}$
            $\boldsymbol{x_i} = vec_{\boldsymbol{\mu_m}}(log_{\boldsymbol{\mu_m}}(\boldsymbol{X_i}))$
        (c) Fit a classifier $T_m(x)$ to the training data using weights $w_i$
        (d) Compute $err_m = \frac{\sum_{i=1}^{N} w_i \cdot 1_{\{y_i \neq T_m(\boldsymbol{x_i})\}}}{\sum_{i=1}^{N} w_i}$
        (e) Compute $\alpha_m = \log(\frac{1-err_m}{err_m}) + \log(K-1)$
        (f) Update weights $w_i = w_i \cdot \exp(\alpha_m \cdot 1_{\{y_i \neq T_m(\boldsymbol{x_i})\}})$ for $i = 1...N$
        (g) Re-normalize weights $w_i$
  Output: $C(x) = \arg\max_k \sum_{m=1}^{M} \alpha_m \cdot 1_{\{T_m(\boldsymbol{x})=k)\}}$
```

It is worth noting that the mean of the points change after each iteration due to the adjustment of the weights. The main idea of Adaboost is that after every boosting iteration (where a new classifier is trained), the weights of misclassified samples get bigger until they are correctly classified, thus making the later classifiers stronger after each iteration. Finally, the output of Adaboost is determined by a majority vote across all classifiers.

3.2 Multi-class Logitboost

Coming from the same family as Adaboost, Logitboost also leverages the idea of utilizing multiple weak learners, but instead it aims to minimize the negative binomial log-likelihood of the samples (see $p_k(x)$ below).

```
Algorithm 2: multi-class Logitboost on Riemannian manifolds
  Input: Training set $(\boldsymbol{X_i}, y_i)_{i=1...N}$, $\boldsymbol{X_i} \in \mathcal{M}$, $y_i \in \{1, 2, ..., K\}$
    1. Initialize weights $w_{ik} = 1/N$, probabilities $p_k(x) = 1/K$ and
       $F_k(x) = 0, i = 1...N, k = 1...K$
    2. Convert labels $y_i$ into one-hot encoded vectors:
       $y_{ik} = \begin{cases} 1 & \text{if } k = y_i, \\ 0 & \text{otherwise} \end{cases}$
    3. Repeat for $m = 1, 2, ..., M$:
       (a) Repeat for $k = 1, 2, ..., K$:
             i. Compute working responses and weights in the k-th class:
                $\boldsymbol{z_{ik}} = \frac{y_{ik} - p_k(x_i)}{p_k(x_i)(1 - p_k(x_i))}$
                $\boldsymbol{w_{ik}} = p_k(x_i)(1 - p_k(x_i))$
            ii. Compute weighted mean of the points
                $\boldsymbol{\mu_{mk}} = \arg\min_{\boldsymbol{X} \in \mathcal{M}} \sum_{i=1}^{N} \boldsymbol{w_{ik}} d^2(\boldsymbol{X_i}, \boldsymbol{X})$
           iii. Map data points to tangent space at $\boldsymbol{\mu_{mk}}$
                $\boldsymbol{x_i} = vec_{\boldsymbol{\mu_{mk}}}(log_{\boldsymbol{\mu_{mk}}}(\boldsymbol{X_i}))$
            iv. Fit $f_{mk}(x)$ by weighted least-squares regression of $\boldsymbol{z_{ik}}$ to $\boldsymbol{x_i}$ with
                weights $\boldsymbol{w_{ik}}$
       (b) Set $f_{mk}(x) \leftarrow \frac{K-1}{K}(f_{mk}(x) - \frac{1}{K} \sum_{k=1}^{K} f_{mk}(x))$ and
           $\boldsymbol{F}_k(x) \leftarrow \boldsymbol{F}_k(x) + f_{mk}(x)$
       (c) Update $p_k(x) = \frac{e^{\boldsymbol{F}_k(x)}}{\sum_{k=1}^{K} e^{\boldsymbol{F}_k(x)}}$ (the softmax operation), $\sum_{k=1}^{K} \boldsymbol{F}_k(x) = 0$
  Output: The classifier $\arg\max_k \boldsymbol{F}_k(x)$
```

One can see that compared to Adaboost, the Logitboost algorithm also contains the steps involving mapping the SPD matrices to vectors using tangent planes, although the order is slightly different. Although the accuracy of Logitboost is generally better than Adaboost, it suffers from computation issues, as the back-and-forth mapping of tangent planes happen for every class, in every iteration, making it very expensive as the number of samples and classes increase. With that in mind, kernel SVM was proposed as the next solution to circumvent this issue.

3.3 Riemannian Gaussian Kernel SVM

This method is not only less computationally expensive than the Boosting algorithms, but it also gives a more accurate representation of distances on the Riemannian manifold (geodesic distance). The kernel function and more specific information can be found in [4], and in the algorithm below:

```
Algorithm 3: Multi-class kernel SVM on Riemannian manifolds
  Input: Training set $(\boldsymbol{X_i}, y_i)_{i=1...N}$, $\boldsymbol{X_i} \in \mathcal{M}$, $y_i \in \{1, 2, ..., K\}$
    1. Determine $\gamma$, either manually or using the below formula (only for the
       Riemannian case):
           $\gamma = N[\sum_{i=0}^{N} d_L^2(\boldsymbol{X_i}, \boldsymbol{\Pi_L}(\{1/N, \boldsymbol{X_i}\}_{1\leq i\leq N}))]^{-1}$
    2. Compute the Gaussian (or RBF) kernel function as:
           $k_G(\boldsymbol{X}, \boldsymbol{Y}) = \exp(-\gamma d_L(\boldsymbol{X}, \boldsymbol{Y}))$
    3. Fit an SVM classifier using $k_G(\boldsymbol{X}, \boldsymbol{Y})$ as the Gram matrix and labels $y_i$
  Output: Prediction of the SVM classifier on the testing set
```

The distance function d_L mentioned above depends on where the data lies, it can be the typical l-norm for the Euclidean case, or it can be the geodesic distance in the Riemannian world:

$$d_L(\boldsymbol{X}, \boldsymbol{Y}) = \|\log(\boldsymbol{X}^{1/2}\boldsymbol{Y}\boldsymbol{X}^{1/2})\|_F$$

In the Riemannian case, there is an extra parameter that needs to be determined to calculate the optimal γ, which is the log-Euclidean mean:

$$\boldsymbol{\Pi_L}(\{\alpha_i, \boldsymbol{X}_i\}_{1\leq i\leq N}) = \exp(\textstyle\sum_{i=1}^{N} \alpha_i \log(\boldsymbol{X}_i))$$

The value of γ determines the extent to which a sample affects the others. According to [1], higher γ values lead to samples having to be closer to affect each other.

4 Methodologies

Since a test set was only available for some of the data-sets (mostly due to the them being in a contest, so the test labels are not given), different methods of benchmarking were used to assert the accuracy of the algorithms. The details are given in Table 1.

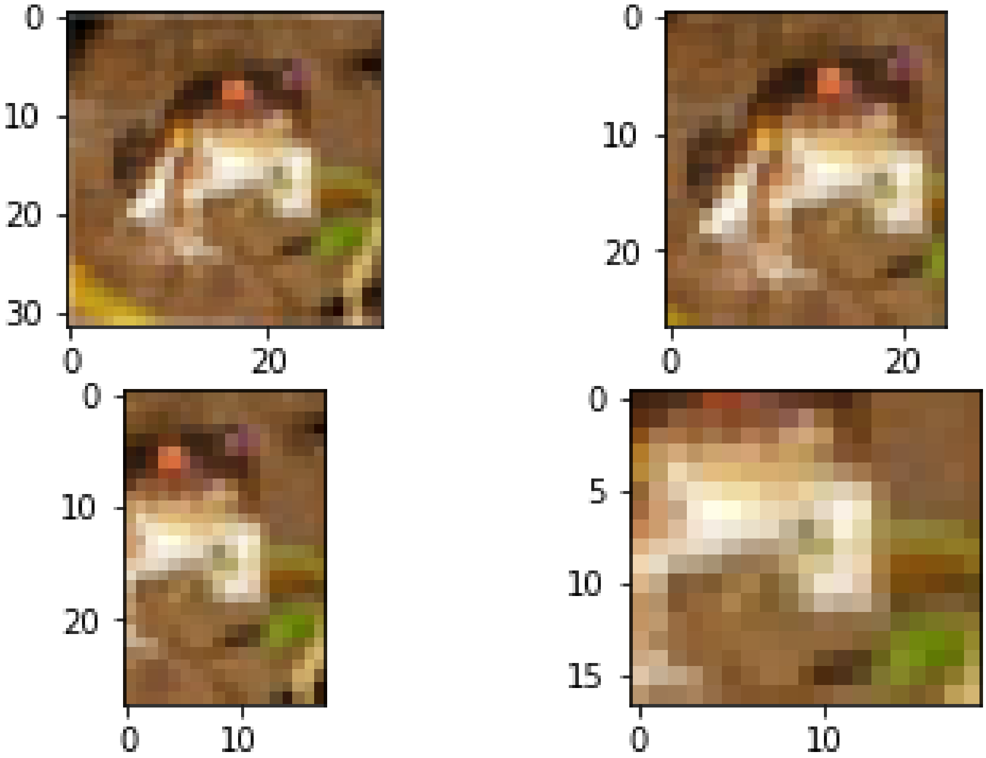
These data-sets were then converted into 8x8 covariance descriptors, which lie in the Riemannian manifold space. To test the methods in the normal Euclidean (vector)

Table 1: Methods used for each data-set

data-set	# Train	# Test	# Classes
CIFAR-10	60000	10000	10
MIO-TCD	39194	5-fold CV	6
TAU-Vehicle	16378	5-fold CV	11

space, the top half of the matrices were flattened (including the diagonal) to a 36-wide vector due to the semi-positive-definiteness nature of the matrices.
Instead of using the whole image, we randomly sample multiple sub-regions of the image, which is then converted into the same covariance descriptors as mentioned above. With this approach, the aim is to generate more samples than just the single training image, and hopefully the sub-regions can have a better focus on the target object than the whole image. A demonstration of this strategy can be seen below.

Figure 4: Sub-region sampling demonstration



After preprocessing the data-sets to produce the Euclidean and Riemannian features, multi-class Logitboost and Adaboost algorithms were applied to both features with 2 different types of base estimator, 2-node and 8-node stumps, respectively. The accuracy is then obtained using the first 50, 100 and 200 iterations of each algorithm. Additonally, multi-class kernel SVM was used as the third algorithm in the experiment. The general idea of the SVM is that the samples will be converted into a Gram matrix containing the ”distance” between one and the rest of the samples. In total, 2 Gram matrices will be made for each run, one comparing the training set with itself, while the other comparing the training with the test set. The only difference in the Riemannian and Euclidean is in how the ”distance” is calculated, with the l2-norm and geodesic distance defined for the Euclidean and Riemannian case respectively.

5 Results

This section presents the results obtained from using the methods mentioned above.

5.1 No sub-region approach

The results can be seen in Table 2. Note that this method is no longer used, so the results can be quite outdated.

Table 2: Accuracy of classification algorithms

			2 Terminal Nodes			**8 Terminal Nodes**		
	Method	**Iterations**	**50**	**100**	**200**	**50**	**100**	**200**
	CIFAR-10							
	Adaboost		0.252	0.279	0.302	0.319	0.342	0.353
E	Logitboost		0.356	0.377	0.394	0.405	0.412	0.397
	RBF SVM		0.417					
	Adaboost		0.286	0.319	0.348	0.355	0.374	0.392
R	Logitboost		0.398	0.420	0.433	0.443	0.444	0.435
	RBF SVM		0.457					
	MIO-TCD							
	Adaboost		0.403	0.417	0.438	0.515	0.528	0.523
E	Logitboost		0.555	0.579	0.595	0.682	0.703	0.717
	RBF SVM		0.372					
	Adaboost		0.434	0.445	0.467	0.530	0.543	0.548
R	Logitboost		0.586	0.611	0.630	0.720	0.741	0.754
	RBF SVM		0.625					
	TAU-Vehicle							
	Adaboost		0.413	0.440	0.414	0.376	0.358	0.370
E	Logitboost		0.533	0.547	0.555	0.541	0.530	0.528
	RBF SVM		0.079					
	Adaboost		0.452	0.425	0.405	0.398	0.397	0.403
R	Logitboost		0.566	0.576	0.584	0.565	0.558	0.555
	RBF SVM		0.269					

Overall, the Riemannian versions of the algorithms seem to perform better than their Euclidean counterpart, hinting that preserving the structure of the SPD matrices might play an important role in training the models. Additionally, increasing the number of iterations generally improves the accuracy of the predictions, but increasing the number of terminal nodes witness contradicting effects. While CIFAR-10 and MIO-TCD benefits ever so slightly from increasing the number of nodes, TAU-Vehicle does not seem to enjoy the increase in terminal nodes, and oftentimes in iterations as well.

The accuracy score for the kernel SVM method is not very good at the moment, possibly due to the choice of γ, or even the kernel itself. To improve the accuracy, one can try to sample multiple subwindows of an image, rather than using the whole image on its own. With this way, the number of available data will greatly increase, leading to more thorough training for the classifiers.

5.2 Sub-region sampling approach

The results for this strategy can be found in Table 3.

Table 3: Accuracy of classification algorithms with sub-region sampling

			2 Terminal Nodes			**8 Terminal Nodes**		
	Method	**Iterations**	**50**	**100**	**200**	**50**	**100**	**200**
	CIFAR-10							
	Adaboost		0.199	0.210	0.218	0.229	0.237	0.240
E	Logitboost		0.237	0.247	0.253	0.258	0.266	0.261
	RBF SVM		0.245					
	Adaboost		0.201	0.214	0.233	0.235	0.245	0.259
R	Logitboost		0.257	0.274	0.292	0.285	0.288	0.288
	RBF SVM		Out of memory (need 298GB)					
	MIO-TCD							
	Adaboost		0.403	0.413	0.417	0.525	0.537	0.550
E	Logitboost		0.550	0.574	0.592	0.685	0.717	0.746
	RBF SVM		0.391					
	Adaboost		0.427	0.430	0.437	0.492	0.489	0.487
R	Logitboost							
	RBF SVM		0.220					
	TAU-Vehicle							
	Adaboost		0.503	0.511	0.520	0.454	0.463	0.458
E	Logitboost		0.550	0.571	0.588			
	RBF SVM		0.203					
	Adaboost		0.503	0.516	0.522	0.530	0.545	0.560
R	Logitboost							
	RBF SVM		0.109					

The first thing to note with this sub-region sampling strategy is that it requires a significantly larger amount of memory, given that the number of images was multiplied by the number of sub-samples we want per image. This makes it much more challenging to run more accurate but computationally expensive algorithms (like Logitboost) without exceeding the 5 days timeout limit on Triton. The same goes for kernel SVM, whose Euclidean algorithm was reduced to a slightly simpler form, possibly contributing to the lower model accuracies when compared with the no sub-sampling approach.

Otherwise, sub-region sampling seems to have mixed effects on the boosting methods, possibly having some sort of dependence with the data-sets as well. For example, while TAU-Vehicle witnessed a considerable increase in accuracy for most boosting experiments, CIFAR-10 saw the complete opposite trend. MIO-TCD, on the other hand, has a fluctuating outcome between the Euclidean and Riemannian methods.

References

[1] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337 – 407, 2000.

[3] Saharon Rosset Ji Zhu, Hui Zou and Trevor Hastie. Multi-class adaboost. *Statistics and Its Interface*, 2(3):349 – 360, 2009.

[4] Ammar Mian, Elias Raninen, and Esa Ollila. A comparative study of supervised learning algorithms for symmetric positive definite features. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 950–954, 2021.